

ipd4300matxpmTES-10

**Defense Information Infrastructure (DII)
Common Operating Environment (COE)**

**Programming Manual (PM)
for the
Textual Observation API Segment (MATXT)
of the
Tactical Environmental Support System Next Century
[TESS(NC)]
Meteorology and Oceanography (METOC) Database**

Document Version 4.3

15 October 1998

**Prepared for:
Naval Research Laboratory
Marine Meteorology Division
Monterey, CA**

**Prepared by:
Integrated Performance Decisions
Middletown, RI**

PRINTED COPY IS UNCONTROLLED AND MAY BE OBSOLETE

ipd4300matxtpmTES-10

Table of Contents

1	SCOPE.....	1
1.1	Identification	1
1.2	System Overview.....	1
2	REFERENCED DOCUMENTS	5
2.1	Government Documents	5
2.2	Non-Government Documents.....	6
3	MATXT OVERVIEW	7
4	SEGMENT DEVELOPMENT	9
4.1	Writing Applications Using the MATXT APIs.....	10
4.1.1	Ingesting a Textual Observation Into the Database	10
4.1.2	Getting a Catalog of Textual Observations in the Database	13
4.1.3	Getting Textual Observations By Query	15
4.1.4	Updating/Deleting Textual Observations By ID	17
4.2	Building Applications on Windows NT Using the MATXT Library.....	22
4.2.1	Makefile for Windows NT Using the Static MATXT Library	22
4.2.2	Makefile for Windows NT Using the Run-Time MATXT Library	23
4.3	Building Application on the HP-UX Using the MATXT Library	24
4.3.1	Makefile for HP-UX Using the Archive MATXT Library	24
4.3.2	Makefile for HP-UX Using the Run-Time MATXT Library	25
4.4	Customizing Segments.....	26
5	NOTES	27
5.1	Glossary of Acronyms.....	27

List of Figures

1-1	TESS(NC) METOC Database Conceptual Organization	3
-----	---	---

(This page intentionally left blank.)

1 SCOPE

1.1 Identification

This Programming Manual (PM) describes the use of the Textual Observation Application Program Interface (API) (MATXT) segment, Version 4.2, of the Tactical Environmental Support System Next Century [TESS(NC)] Meteorology and Oceanography (METOC) Database. The MATXT segment provides APIs for the storage and retrieval of textual METOC observations and bulletins. This software is designed to run under the Defense Information Infrastructure (DII) Common Operating Environment (COE), release 3.1, on a Hewlett-Packard computer running HP-UX 10.20 or a personal computer running the Microsoft Windows NT 4.0 operating system with Service Pack 3.

1.2 System Overview

The software described in this document forms a portion of the METOC Database component of the TESS(NC) Program (Navy Integrated Tactical Environmental Subsystem (NITES) Version I). On 29 October 1996, the Oceanographer of the Navy issued a TESS Program Policy statement in letter 3140 Serial 961/6U570953, modifying the Program by calling for five seamless software versions that are DII COE compliant, preferably to level 5.

The five versions are:

- NITES Version I The local data fusion center and principal METOC analysis and forecast system (TESS(NC))
- NITES Version II The subsystem on the Joint Maritime Command Information System (JMCIS) or Global Command and Control System (GCCS) (NITES/Joint METOC Segment (JMS))
- NITES Version III The unclassified aviation forecast, briefing, and display subsystem tailored to Naval METOC shore activities (currently satisfied by the Meteorological Integrated Data Display System (MIDDS))
- NITES Version IV The Portable subsystem composed of independent PCs/workstations and modules for forecaster, satellite, communications, and Integrated Command, Control, Communications, Computer, and Intelligence Surveillance Reconnaissance (IC4ISR) functions (currently the Interim Mobile Oceanographic Support System (IMOSS))
- NITES Version V Foreign Military Sales (currently satisfied by the Allied Environmental Support System (AESS))

NITES I acquires and assimilates various METOC data for use by US Navy and Marine Corps weather forecasters and tactical planners. NITES I provides these users with METOC data, products, and applications necessary to support the warfighter in tactical operations and decision making. NITES I provides METOC data and products to NITES I and II applications, as well as non-TESS(NC) systems requiring METOC data, in a heterogeneous, networked computing environment.

The TESS(NC) Concept of Operations and system architecture require that the METOC Database be distributed both in terms of application access to METOC data and products and in terms of physical location of the data repositories. The organizational structure of the database is influenced by these requirements, and the components of this distributed database are described below.

In accordance with DII COE database concepts, the METOC Database is composed of six DII COE-compliant *shared database* segments. Associated with each shared database segment is an API segment. The segments are arranged by data type as follows:

<u>Data Type</u>	<u>Data Segment</u>	<u>API Segment</u>
Grid Fields	MDGRID	MAGRID
Latitude-Longitude-Time (LLT) Observations	MDLLT	MALLT
Textual Observations and Bulletins	MDTXT	MATXT
Remotely Sensed Data	MDREM	MAREM
Imagery	MDIMG	MAIMG
Climatology Data	MDCLIM	MACLIM

A typical client-server installation is depicted in Figure 1-1 on the next page. This shows the shared database segments residing on a DII COE SHADE database server, with a NITES I or II client machine hosting the API segments. Communication between API segments and shared database segments is accomplished over the network using ANSI-standard Structured Query Language (SQL).

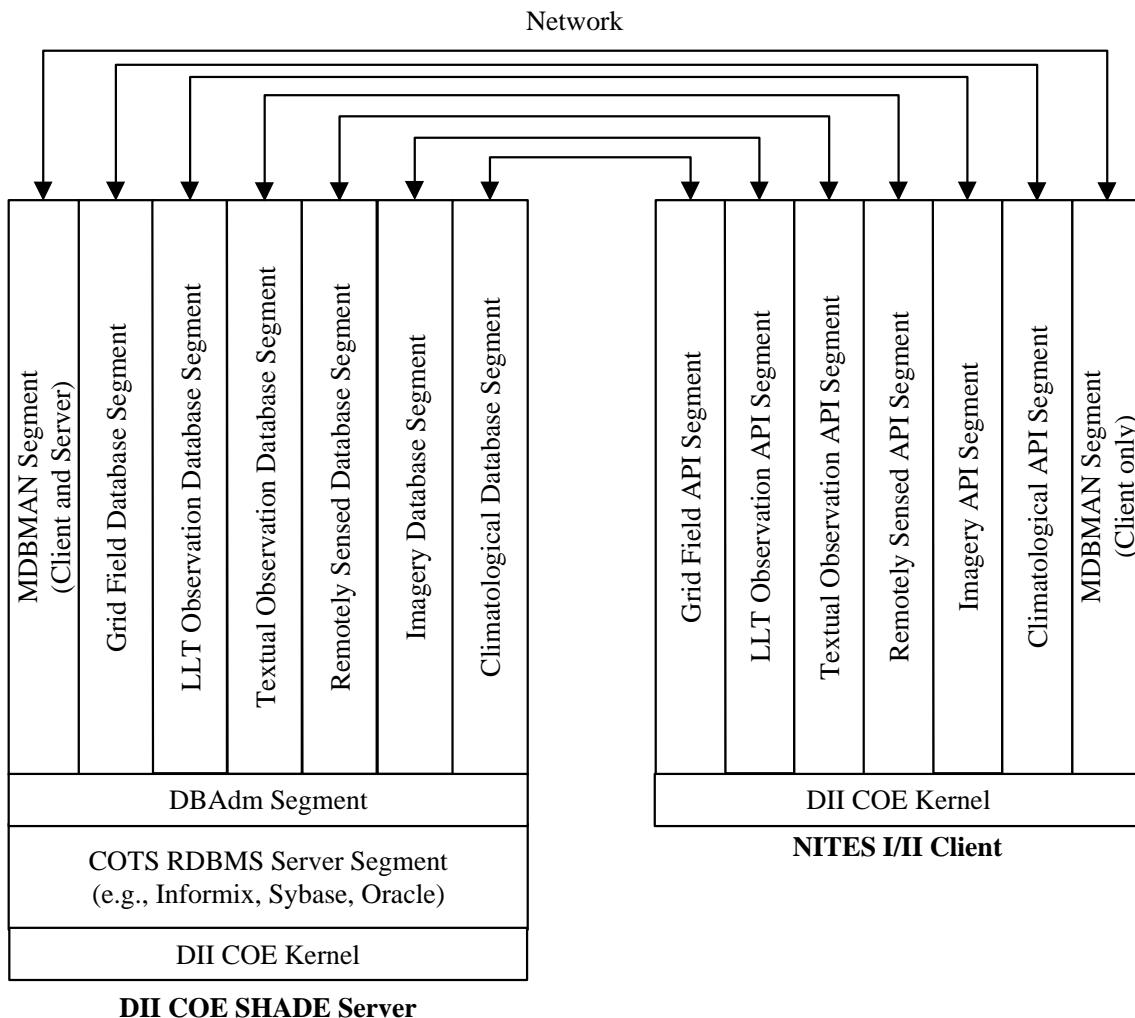


Figure 1-1. TESS(NC) METOC Database Conceptual Organization

The MATXT segment deals with textual observations and bulletins. Textual observational data are primarily ASCII-formatted forecasts or bulletin/warning-oriented messages. Textual observation data can be associated with a specific geographic point and time, but more generally are associated with a geographical area or region. Types include Forecast Reports, Warnings, and Notices. Depending on the type of textual observation, the reporting station or organization and the area or region affected are decoded and stored along with the textual portion of the message. Textual observation data are typically displayed as text by a client application.

(This page intentionally left blank.)

2 REFERENCED DOCUMENTS

2.1 Government Documents

STANDARDS

MIL-STD-498 *Software Development and Documentation*
5 December 1994

SPECIFICATIONS

Unnumbered *Performance Specification (PS) for the Tactical Environmental Support System/Next Century TESS(3)/NC (AN/UMK-3)*
5 December 1997

Unnumbered *Software Requirements Specification for the Tactical Environmental Support System/Next Century [TESS(3)/NC] Meteorological and Oceanographic (METOC) Database*, Space and Naval Warfare Systems Command, Environmental Systems Program Office (SPAWAR PMW-185), Washington, DC
30 September 1997

OTHER DOCUMENTS

Unnumbered *Database Design Description for the Tactical Environmental Support System/Next Century [TESS(3)/NC] Meteorological and Oceanographic (METOC) Database*, Space and Naval Warfare Systems Command, Environmental Systems Program Office (SPAWAR PMW-185), Washington, DC
30 September 1997

DII.COE.DocReqs-5 *Defense Information Infrastructure (DII) Common Operating Environment (COE) Developer Documentation Requirements, Version 1.0*
29 April 1997

ipd4300matxrmTES-10 *Application Program Interface Reference Manual (APIRM) for the Textual Observation API (MATXT) Segment of the Tactical Environmental Support System Next Century [TESS(NC)] Meteorology and Oceanography (METOC) Database*
15 October 1998

ipd4300matxtipTES-10 *Installation Procedures (IP) for the Textual Observation API (MATXT) Segment of the Tactical Environmental Support System Next Century [TESS(NC)] Meteorology and Oceanography (METOC) Database*
15 October 1998

ipd4200matxtsvdTES-10 *Software Version Description (SVD) for the Textual Observation API (MATXT) Segment of the Tactical Environmental Support System Next Century [TESS(NC)] Meteorology and Oceanography (METOC) Database*
15 October 1998

2.2 Non-Government Documents

World Meteorological Organization, Geneva, Switzerland

WMO-306 *Manual On Codes*
1995

3 MATXT OVERVIEW

The MATXT segment provides APIs used to access textual observation and bulletin data in the TESS(NC) METOC Database. The schema for storing these data is defined by the shared database segment MDTXT. Both of these segments require the Informix Relational Database Management System (RDBMS), version 7.22 (for HP-UX machines) or 7.23 (for Windows NT machines). Both segments run in the DII COE release 3.1, hosted on the following machines and operating systems:

- Tactical Advanced Computer, TAC-3 (HP 750/755)/TAC-4 (HP J210) Operating System: HP-UX 10.20
- IBM-Compatible Personal Computer (PC) Operating System: Microsoft Windows NT 4.0, with Service Pack 3

MATXT uses the following environment variables related to the Informix installation:

- **INFORMIXSERVER** Identifies the Informix server, typically set to `online_coe`
- **INFORMIXDIR** Path to the Informix software, typically `/opt/informix` on HP systems and `C:\informix` on Windows NT systems

The system's PATH should include the directory `$(INFORMIXDIR)/bin` for HP-UX systems and `$(INFORMIXDIR)\bin` for Windows NT systems.

The MATXT segment is delivered as both archive and runtime libraries on both platforms, with filenames as follows:

<u>Library Type</u>	<u>HP-UX Filename</u>	<u>Windows NT Filename</u>
Archive	<code>libMATXTAPI.a</code>	<code>matxtapi.lib</code>
Runtime	<code>libMATXTAPI.sl</code>	<code>matxtapi.dll</code>

The runtime libraries are typically installed in the directory `/h/MATXT/bin` on HP systems and `C:\h\MATXT\bin` on Windows NT systems. On HP-UX systems, `/h/MATXT/bin` must be added to the `SHLIB_PATH` environment variable. To use the runtime libraries in Windows NT, `C:\h\MATXT\bin` must be in the user's PATH, and `_MDBDLL` must be defined. This `_MDBDLL` can be defined in different places. It could be defined in the source code (`#define _MDBDLL`), in a Makefile.nt (`DEFINES = -D_MDBDLL`), or under project settings in MS Visual Studio.

The archive libraries are typically installed in the directory /h/MATXT/lib on HP systems and C:\h\MATXT\lib on Windows NT systems.

The individual API methods are described in the APIRM referenced in Section 2. Section 4 of this document provides instructions and programming examples for the use of the APIs.

4 SEGMENT DEVELOPMENT

Programming applications to access textual observation and bulletin data is straightforward. The Textual Observation API segment provides interfaces to:

- Connect to the TESS(NC) MDTXT Database (MATXTConnect, MATXTRemoteConnect)
- Set the current database connection (MATXTSetConnection)
- Ingest a textual observation into the database (MATXTIngest)
- Retrieve a catalog listing of textual observations meeting specified criteria from the database (MATXTCatalog)
- Retrieve selected textual observation data from the database (MATXTGetByQuery)
- Retrieve a single textual observation from the database (MATXTGetByID)
- Update a textual observation data record in the database (MATXTUpdateByID)
- Delete a textual observation data record from the database (MATXTDeleteByID, MATXTDeleteByQuery)
- Free the linked lists returned by MATXTCatalog and MATXTGetByQuery (MATXTFreeLL)
- Disconnect from the database (MATXTDisconnect, MATXTRemoteDisconnect).

Each of these methods is described in detail in the MATXT APIRM, referenced in Section 2.

4.1 Writing Applications Using the MATXT APIs

This section shows the use of the MATXT APIs to perform common data access tasks. In each case, an overview of the actions to be performed is provided, along with a code example and a discussion of pertinent programming concerns.

In all cases, note that the MATXTConnect method must be called to connect the application to the database before any other operations may be performed. MATXTDisconnect should be called to disconnect the application from the database at the end of the session. These methods only need to be called once per session.

All structures must be initialized through use of **calloc** or **memset**. Failure to initialize a structure could cause unpredictable results. Garbage in a structure could cause a query to fail.

The MATXTRET structure is used to return status information from each MATXT method. See Section 3.1.6 of the APIRM for details of the information returned in this structure.

4.1.1 Ingesting a Textual Observation Into the Database

The MATXTIngest method is used to add a textual observation record to the database. It takes as input a pointer to a MATXTObs structure containing the metadata for the observation and the observation to be added. All data in the MATXTObs structure should be filled in. If the observation does not cover a defined area, the boundaries in the *matxtAOI* element should be set to reflect a global area (i.e., latitude -90.0 to 90.0, longitude -180.0 to 180.0).

The code below provides an example of the procedure for adding observations to the database.

```
*****
*
* MATXT_IngestSample - This program is a driver that will add an
*                      AIRMET to the MDTXT database.
*
* The flow is as follows
*   - Populate the MATXTOBS structure with the AIRMET to ingest
*   - Invoke the subroutine MATXTConnect to connect to the db.
*   - Invoke the subroutine MATXTIngest to ingest the data.
*   - Invoke the subroutine MATXTDisconnect to disconnect from
*     the db.
*
*****
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <stdlib.h>
#include "MATXTAPI.h"      /* All of the MATXT structures and defines are */
                           /* included in here */
```

```
void main ( )
{
    MATXTRET matxtRet;          /* This is returned from all MATXT subroutines */
    MATXTOBS matxtObs;         /* Main textual observation structure */
    int      nObjectID = 0;     /* Object ID of the ingested data */
    char     szString[512];     /* Temporary string.. */

    /*****
     * Initialize the structures. Do this so that there is no garbage
     *****/
    memset ( &matxtRet, 0, sizeof ( MATXTRET ) );
    memset ( &matxtObs, 0, sizeof ( MATXTOBS ) );

    /*****
     * Set up the type and subtype fields according to manual
     *****/
    matxtObs.chType = 'W';
    matxtObs.chSubType = 'A';

    /*****
     * Set the quality indicator to zero to indicate good data
     *****/
    matxtObs.nQualityIndicator = 0;

    /*****
     * For this sample we will just set the report time to the current
     *****/
    matxtObs.lReportTime = ( long ) time ( NULL );

    /*****
     * For this sample we will just set the valid times to zero which
     * means unknown.
     *****/
    matxtObs.lBegValidTime = 0;
    matxtObs.lEndValidTime = 0;

    /*****
     * Just leave receipt time at zero. DB will fill it in.
     *****/
    matxtObs.lReceiptTime = 0;

    /*****
     * For this sample we will just hard code in some lat / lons
     *****/
    matxtObs.matxtAOI.rsNorthLat = 50.0;
    matxtObs.matxtAOI.rsSouthLat = 40.0;
    matxtObs.matxtAOI.rsEastLon = -110.0;
    matxtObs.matxtAOI.rsWestLon = -140.0;

    /*****
     * Set data category to zero, meaning base
     *****/
    matxtObs.nDataCategory = 0;

    /*****
     * Set up the classification level
     *****/
    strcpy ( matxtObs.szSecurityClass, "UNCLASSIFIED" );

    /*****
     * Set the originating site
     *****/
```

```
*****
strcpy ( matxtObs.szOriginatingSite, "KMRY" );

*****
* Set the receipt method ( how you got the data )
*****
strcpy ( matxtObs.szReceiptMethod, "COMEDS" );

*****
* This is the actual message
*****
strcpy ( szString,
"WAUS1 KDFW 300245\n \
DFWS WA 300245\n \
AIRMET SIEA FSOD WITH JTST. CONDS CONTG BYD 09Z THRU 15Z AND MOVG EWD.\n \
....;\n \
DFWZ WA 300245\n \
AIRMET ZULU FOR ICE AND FRZLVL VALID UNTIL 300900\n \
.\n \
AIRMET ICE...TX LA MS AL CSTL WTRS\n \
FROM AEX TO 40W CE RK AO2;\n \
\n \
\n \
NNNN\n " );

*****
* Set the data size and memcpy the message into the pData field
*****
matxtObs.nDataSize = strlen ( szString );
matxtObs.pData = ( char * ) calloc ( sizeof ( char ), matxtObs.nDataSize );
memcpy ( matxtObs.pData, szString, matxtObs.nDataSize );

*****
* Connect to the database. Notice how the status is displayed.
*****
matxtRet = MATXTConnect ( );
printf ( "After MATXTConnect: %d:%s:%s\n", matxtRet.nStatus,
        matxtRet.szSQLState, matxtRet.szErrorMessage );

*****
* Call the subroutine to ingest the data
*****
if ( !matxtRet.nStatus )
{
    matxtRet = MATXTIngest ( &matxtObs, &nObjectID );
    printf ( "After MATXTIngest: %d:%s:%s\n", matxtRet.nStatus,
            matxtRet.szSQLState, matxtRet.szErrorMessage );
    if ( !matxtRet.nStatus )
        printf ( "Added ObjectID %d\n", nObjectID );
}

*****
* Disconnect from the database
*****
matxtRet = MATXTDisconnect ( );
printf ( "After MATXTDisconnect: %d:%s:%s\n", matxtRet.nStatus,
        matxtRet.szSQLState, matxtRet.szErrorMessage );

} /* End of main */
```

4.1.2 Getting a Catalog of Textual Observations in the Database

The MATXTCatalog method is used to retrieve a catalog of observations in the database that meet specific criteria. The criteria are contained in the input MATXTCATQUERY structure. Most fields in this structure are optional; the only requirement is for the *matxtAOI* element to be filled in with the area of interest boundaries. These may be set to global (latitude -90.0 to 90.0, longitude -180.0 to 180.0) if no specific area is desired. MATXTCatalog returns a linked list of MATXTCATDATA structures containing the metadata for each record found that met the input criteria. It also returns the number of matching records found and the MATXTRET status structure.

The linked list returned by MATXTCatalog must be freed with a call to MATXTFreeLL when no longer needed, and the head of the linked list must then be freed also.

The code below provides an example showing the use of the MATXTCatalog method.

```
*****
*
*  MATXT_CatalogSample - This program is a driver that will retrieve *
*                      a catalog listing of all textual observation *
*                      warnings from the database
*
*  The flow is as follows
*      - Populate the MATXTQUERY structure
*      - Invoke the subroutine MATXTConnect to connect to the db.
*      - Invoke the subroutine MATXTCatalog to retrieve the listing
*      - Loop through the linked list printing out information
*      - Invoke the subroutine MATXTDisconnect to disconnect from
*          the db.
*      - Invoked MATXTFreeLL to the free the linked list returned
*          from MATXTCatalog
*
*****
*/
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <stdlib.h>
#include "MATXTAPI.h"      /* All of the MATXT structures and defines are */
                           /* included in here */

void main ( )
{
    MATXTRET matxtRet;           /* This is returned from all MATXT */
                               /* subroutines */
    MATXTCATQUERY matxtCatQuery; /* Catalog query criteria */
    MATXTCATDATA *pmatxtCatData; /* Returned from the catalog retrieve */
    MATXTLINKEDLIST matxtLL;    /* Linked list, returned from catalog */
    MATXTLINKEDLIST *pmatxtLoop; /* Linked list, returned from catalog */
    int nNumFound = 0;           /* Number found from catalog */

```

```

 ****
 * Initialize the structures.  Do this so that there is no garbage      *
 ****
memset ( &matxtRet, 0, sizeof ( MATXTRET ) );
memset ( &matxtCatQuery, 0, sizeof ( MATXTCATQUERY ) );
memset ( &matxtLL, 0, sizeof ( MATXTLINKEDLIST ) );

 ****
 * Fill in the matxt catalog query structure                         *
 ****
/* We want to retrieve all warnings                                     */
matxtCatQuery.chType = 'W';

 ****
 * That is, all warnings from the last 10 days                      *
 ****
matxtCatQuery.lEndReportTime = ( long ) time ( NULL );
matxtCatQuery.lBegReportTime = matxtCatQuery.lEndReportTime
                           - ( 10 * 24 * 60 * 60 );
/* For the world                                                       */
matxtCatQuery.matxtAOI.rsNorthLat = 90.0;
matxtCatQuery.matxtAOI.rsSouthLat = -90.0;
matxtCatQuery.matxtAOI.rsEastLon = 180.0;
matxtCatQuery.matxtAOI.rsWestLon = -180.0;

 ****
 * Connect to the database.  Notice how the status is displayed       *
 ****
matxtRet = MATXTConnect ( );
printf ( "After MATXTConnect: %d:%s:%s\n", matxtRet.nStatus,
        matxtRet.szSQLState, matxtRet.szErrorMessage );

 ****
 * Retrieve the catalog listing of text ob warnings from the          *
 * last 10 days                                                       *
 ****
matxtRet = MATXTCatalog ( &matxtCatQuery, &nNumFound, &matxtLL );
printf ( "After MATXTCatalog: %d:%s:%s\n", matxtRet.nStatus,
        matxtRet.szSQLState, matxtRet.szErrorMessage );
printf ( "MATXTCatalog found %d entries\n", nNumFound );

 ****
 * Loop through the linked list displaying some of the information   *
 ****
for ( pmatxtLoop = ( MATXTLINKEDLIST * ) &matxtLL;
      pmatxtLoop != NULL && nNumFound > 0;
      pmatxtLoop = ( MATXTLINKEDLIST * ) pmatxtLoop->pNext )
{
    pmatxtCatData = ( MATXTCATDATA * ) pmatxtLoop->pData;
    printf ( "\tnObjectID = %d\n", pmatxtCatData->nObjectID );
    printf ( "\tchType = %c chSubType = %c\n", pmatxtCatData->chType,
            pmatxtCatData->chSubType );
    printf ( "\tlReportTime = %d \n\t\t\t\t%s", pmatxtCatData->lReportTime,
            ctime ( &pmatxtCatData->lReportTime ) );
    printf ( "\tNorth = %.2f, South = %.2f, East = %.2f, West = %.2f\n",
            pmatxtCatData->matxtAOI.rsNorthLat,
            pmatxtCatData->matxtAOI.rsSouthLat,
            pmatxtCatData->matxtAOI.rsEastLon,
            pmatxtCatData->matxtAOI.rsWestLon );
}

```

```
pmatxtCatData->matxtAOI.rsSouthLat,
pmatxtCatData->matxtAOI.rsEastLon,
pmatxtCatData->matxtAOI.rsWestLon );
printf ( "\tReceipt Method = %s\n", pmatxtCatData->szReceiptMethod );
printf ( "+++++++\n" );
}

/*****************
 * Free up the linked list
*****************/
matxtRet = MATXTFreeLL ( &matxtLL );
printf ( "After MATXTFreeLL: %d:%s:%s\n", matxtRet.nStatus,
matxtRet.szSQLState, matxtRet.szErrorMessage );

/*****************
 * Disconnect from the database
*****************/
matxtRet = MATXTDisconnect ( );
printf ( "After MATXTDisconnect: %d:%s:%s\n", matxtRet.nStatus,
matxtRet.szSQLState, matxtRet.szErrorMessage );

} /* End of main */
```

4.1.3 Getting Textual Observations By Query

The MATXTGetByQuery method is used to retrieve textual observations from the database based upon a criterion. The criteria are contained in the input MATXTCATQUERY structure. Most fields in this structure are optional; the only requirement is for the *matxtAOI* element to be filled in with the area of interest boundaries. These may be set to global (latitude -90.0 to 90.0, longitude -180.0 to 180.0) if no specific area is desired. MATXTGetByQuery returns a linked list of MATXTOBS structures containing the data for each record found that met the input criteria. It also returns the number of matching records found and the MATXTRET status structure.

The linked list returned by MATXTCatalog must be freed with a call to MATXTFreeLL when no longer needed, and the head of the linked list must then be freed also.

The code below provides an example showing the use of the MATXTGetByQuery method.

```
*****
*
* MATXT_GetByQuerySample - This program is a driver that will
* retrieve textual observations from the
* database based upon a criteria
*
* The flow is as follows
*   - Populate the MATXTQUERY structure
*   - Invoke the subroutine MATXTConnect to connect to the db.
*   - Invoke the subroutine MATXTGetByQuery to retrieve the obs
*   - Loop through the linked list printing out information
*   - Invoke the subroutine MATXTDisconnect to disconnect from
*     the db.
*
```

```
*      - Invoked MATXTFreeLL to free the linked list returned      *
*      from MATXTCatalog                                         *
*      *                                                 *
*      *                                                 *
******/
```

```
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <stdlib.h>
#include "MATXTAPI.h"      /* All of the MATXT structures and defines are */
/* included in here */
```

```
void main ( )
{
    MATXTRET matxtRet;          /* This is returned from all MATXT      */
                               /* subroutines */
```

```
    MATXTCATQUERY matxtCatQuery; /* Catalog query criteria */
```

```
    MATXTLINKEDLIST matxtLL;   /* Linked list, returned from catalog */
```

```
    MATXTLINKEDLIST *pmatxtLoop; /* Linked list, returned from catalog */
```

```
    MATXTOBS      *pmatxtObs;  /* Textual observation structure */
```

```
    int    nNumFound = 0;       /* Number found from retrieval */
```

```
    *****/
    /* Initialize the structures. Do this so that there is no garbage */
    *****/
    memset ( &matxtRet, 0, sizeof ( MATXTRET ) );
    memset ( &matxtCatQuery, 0, sizeof ( MATXTCATQUERY ) );
    memset ( &matxtLL, 0, sizeof ( MATXTLINKEDLIST ) );
```

```
    *****/
    /* Fill in the matxt catalog query structure */
    *****/

```

```
    *****/
    /* We want to retrieve all notices */
    *****/
    matxtCatQuery.chType = 'N';

    *****/
    /* That is, all notices from the last 10 days */
    *****/
    matxtCatQuery.lEndReportTime = ( long ) time ( NULL );
    matxtCatQuery.lBegReportTime = matxtCatQuery.lEndReportTime
                                - ( 10 * 24 * 60 * 60 );

    *****/
    /* For the world */
    *****/
    matxtCatQuery.matxtAOI.rsNorthLat = 90.0;
    matxtCatQuery.matxtAOI.rsSouthLat = -90.0;
    matxtCatQuery.matxtAOI.rsEastLon = 180.0;
    matxtCatQuery.matxtAOI.rsWestLon = -180.0;

    *****/
    /* Connect to the database. Notice how the status is displayed */
    *****/
    matxtRet = MATXTConnect ( );
    printf ( "After MATXTConnect: %d:%s:%s\n", matxtRet.nStatus,
            matxtRet.szSQLState, matxtRet.szErrorMessage );
```

```
    *****/
```

```
* Retrieve the textual observations meeting the criteria we set      *
*****  
*****  
matxtRet = MATXTGetByQuery ( &matxtCatQuery, &nNumFound, &matxtLL );
printf ( "After MATXTGetByQuery: %d:%s:%s\n", matxtRet.nStatus,
        matxtRet.szSQLState, matxtRet.szErrorMessage );
printf ( "MATXTGetByQuery found %d entries\n", nNumFound );  
  
*****  
* Loop through the linked list displaying some of the information  *
*****  
for ( pmatxtLoop = ( MATXTLINKEDLIST * ) &matxtLL;
      pmatxtLoop != NULL && nNumFound > 0;
      pmatxtLoop = ( MATXTLINKEDLIST * ) pmatxtLoop->pNext )
{
    pmatxtObs = ( MATXTOBS * ) pmatxtLoop->pData;
    printf ( "\tchType = %c chSubType = %c\n", pmatxtObs->chType,
            pmatxtObs->chSubType );
    printf ( "\tlReportTime = %d \n\t\t\t\t%s", pmatxtObs->lReportTime,
            ctime ( &pmatxtObs->lReportTime ) );
    printf ( "\tNorth = %.2f, South = %.2f, East = %.2f, West = %.2f\n",
            pmatxtObs->matxtAOI.rsNorthLat,
            pmatxtObs->matxtAOI.rsSouthLat,
            pmatxtObs->matxtAOI.rsEastLon,
            pmatxtObs->matxtAOI.rsWestLon );
    printf ( "\tReceipt Method = %s\n", pmatxtObs->szReceiptMethod );
    printf ( "+++++++\n" );
}  
  
*****  
* Free up the linked list
*****  
matxtRet = MATXTFreeLL ( &matxtLL );
printf ( "After MATXTFreeLL: %d:%s:%s\n", matxtRet.nStatus,
        matxtRet.szSQLState, matxtRet.szErrorMessage );  
  
*****  
* Disconnect from the database
*****  
matxtRet = MATXTDisconnect ( );
printf ( "After MATXTDisconnect: %d:%s:%s\n", matxtRet.nStatus,
        matxtRet.szSQLState, matxtRet.szErrorMessage );  
  
} /* End of main */
```

4.1.4 Updating/Deleting Textual Observations By ID

The MATXTUpdateByID method is used to modify a textual observation that exists in the database. Any field in the MATXTOBS structure can be updated. The MATXTUpdateByID method takes as input the object ID, found from MATXTCatalog, and the MATXTOBS structure. It returns the MATXTRET status structure.

The MATXTDeleteByID method is used to remove a textual observation from the database. It takes as input the object ID, found from MATXTCatalog, and returns the MATXTRET status structure.

The MATXTGetByID method is used to retrieve a single textual observation from the database. It takes as input the objectID, found from MATXTCatalog. It returns the MATXTOBS structure, as well as the MATXTRET status structure.

The following example will invoke the MATXTCatalog method to retrieve all warnings. The example will then loop through the returned linked list dereferencing the pData field of the linked list into a MATXTOBS structure. The code will then check the chSubType field. All AIRMETS will be retrieved by a call to MATXTGetByID and then updated by a call to MATXTUpdateByID. All SIGMETS will be deleted by a call to MATXTDeleteByID.

The code below provides an example showing the use of the MATXTUpdateByID method.

```
*****
*
*   MATXT_CatalogSample - This program is a driver that will retrieve      *
*                         a catalog listing of all textual observation      *
*                         warnings from the database. Based upon the       *
*                         type/subtype of the ob various actions are      *
*                         taken.                                         *
*
*   The flow is as follows
*     - Populate the MATXTQUERY structure
*     - Invoke the subroutine MATXTConnect to connect to the db.
*     - Invoke the subroutine MATXTCatalog to retrieve the listing
*     - Loop through the linked list printing out information
*       - For each entry check the Type field
*         - if SubType is AIRMET then retrieve and update it using
*           MATXTGetByID and MATXTUpdateByID
*         - if SubType is SIGMET then delete it using
*           MATXTDeleteByID
*     - Invoke the subroutine MATXTDisconnect to disconnect from
*       the db.
*     - Invoked MATXTFreeLL to the free the linked list returned
*       from MATXTCatalog
*
*****
*/
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <stdlib.h>
#include "MATXTAPI.h"      /* All of the MATXT structures and defines are */
                           /* included in here */

void main ( )
{
    MATXTRET matxtRet;          /* This is returned from all MATXT
                                  /* subroutines
    MATXTCATQUERY matxtCatQuery; /* Catalog query criteria
    MATXTCATDATA *pmatxtCatData; /* Returned from the catalog retrieve
    MATXTLINKEDLIST matxtLL;    /* Linked list, returned from catalog
    MATXTLINKEDLIST *pmatxtLoop; /* Linked list, returned from catalog
    MATXTOBS matxtObs;          /* Textual observation structure
    int nNumFound = 0;          /* Number found from catalog
```



```
        ctime ( &pmatxtCatData->lReportTime ) );
printf ( "\tNorth = %5.2f, South = %5.2f, East = %5.2f, West = %5.2f\n",
        pmatxtCatData->matxtAOI.rsNorthLat,
        pmatxtCatData->matxtAOI.rsSouthLat,
        pmatxtCatData->matxtAOI.rsEastLon,
        pmatxtCatData->matxtAOI.rsWestLon );
printf ( "\tReceipt Method = %s\n", pmatxtCatData->szReceiptMethod );

/*********************  
* Check and see if its a AIRMET  
*****  

if ( pmatxtCatData->chSubType == 'A' )
{
    printf ( "\t\t***Found an AIRMET***\n" );
    /* Retrieve it using MATXTGetByID  
*****  

    matxtRet = MATXTGetByID ( pmatxtCatData->nObjectID, &matxtObs );
    printf ( "\t\t\tAfter MATXTGetByID: %d:%s:%s\n",
            matxtRet.nStatus,
            matxtRet.szSQLState, matxtRet.szErrorMessage );
    if ( ! matxtRet.nStatus )
    {
        printf ( "\t\t\tMessage is '%s'\n", matxtObs.pData );
        matxtObs.nDataCategory = 2; /* 2 Means edited.. */
        /* Now update it using MATXTUpdateByID  
*****  

        matxtRet = MATXTUpdateByID ( &matxtObs, &pmatxtCatData->nOb );
        printf ( "\t\t\tAfter MATXTUpdateByID: %d:%s:%s\n",
            matxtRet.nStatus, matxtRet.szSQLState,
            matxtRet.szErrorMessage );

        /* Now free the message part of the structure
        *****
        free ( matxtObs.pData );
    }
}
/* End of if its an AIRMENT */

/*********************  
* Check and see if its a SIGMET  
*****  

if ( pmatxtCatData->chSubType == 'S' )
{
    printf ( "\t\t***Found an SIGMET***\n" );
    /* Delete it using MATXTDeleteByID  
*****  

    matxtRet = MATXTDeleteByID ( pmatxtCatData->nObjectID );
    printf ( "\t\t\tAfter MATXTDeleteByID: %d:%s:%s\n",
            matxtRet.nStatus, matxtRet.szSQLState,
            matxtRet.szErrorMessage );
}
printf ( "+++++++\n" );
/* Free up the linked list
*****
matxtRet = MATXTFreeLL ( &matxtLL );
```

```
printf ( "After MATXTFreeLL: %d:%s:%s\n", matxtRet.nStatus,
        matxtRet.szSQLState, matxtRet.szErrorMessage );

/* **** Disconnect from the database **** */
matxtRet = MATXTDisconnect ( );
printf ( "After MATXTDisconnect: %d:%s:%s\n", matxtRet.nStatus,
        matxtRet.szSQLState, matxtRet.szErrorMessage );

} /* End of main */
```

4.2 Building Applications on Windows NT Using the MATXT Library

This section contains two makefiles that can be used to build applications that link into the MATXT library. The first makefile links into the static version of the MATXT library. The second makefile links into the run-time MATXT library. The main differences in the two makefiles are the definitions of the CFLAGS and LDFLAGS and the location of the MATXT library. These parts of the makefiles will be in bold to stand out.

4.2.1 Makefile for Windows NT Using the Static MATXT Library

```
#####
#
#   makefile.nt.lib - Builds the sample MATXT_CatalogSample
#
#   Note that this makefile uses cl and link instead of esql for building.
#   You can use either.
#
#   This builds 1 executable:
#           MATXT_CatalogSample_1 - Built with the MATXT static library
#
#####
#
CFILES1 = \
    MATXT_CatalogSample.c

OFILES1 = $(CFILES1:.c=.obj)

MATXT_HOME = c:\h\MATXT

# The program built with Static Library
PROG1_LIB_NAME = MATXT_CatalogSample_1

# The informix libraries
INFORMIXDIR = c:\informix
INFORMIXLIBS = $(INFORMIXDIR)\lib\isqlt07c.lib

LIBS = $(INFORMIXLIBS) $(MATXT_HOME)\lib\MATXTAPI.lib

# MATXT include directory
INCLUDES = -I$(MATXT_HOME)\include -I$(INFORMIXDIR)\incl\esql

# other definitions
RM = -@del /Q /F /S
LD = link

CFLAGS = -c -nologo -G5 -Od -Zi -W3 -D_DEBUG -MT -D_MT $(INCLUDES) \
    -DSTRICT -D__STDC__=0
LDFLAGS = -nologo -pdb:none /VERBOSE:LIB -nologo -pdb:none \
    /NODEFAULTLIB:msvcrt.lib /NODEFAULTLIB:LIBC.LIB

all: clean $(PROG1_LIB_NAME)

$(PROG1_LIB_NAME) : $(OFILES1)
    $(LD) $(LDFLAGS) $(OFILES1) \
```

```
$(LIBS) -OUT:$(PROG1_LIB_NAME).exe
@echo "done"

#      Clean up objects, uids, and executables
clean :
    $(RM) $(OFILES1) *.lnk *.map

clobber: clean
    $(RM) $(PROG1_LIB_NAME).exe
```

4.2.2 Makefile for Windows NT Using the Run-Time MATXT Library

Please note that this makefile uses a `-D_MDBDLL` on its `CFLAGS` line. In order to use the MATXT DLL, `_MDBDLL` needs to be defined somewhere. It can be defined in the makefile (this example), in the source code, or under project settings in MS Visual Studio.

```
#####
#
#  makefile.nt - Builds the sample MATXT_CatalogSample
#
#  Note that this makefile uses cl and link instead of esql for building.
#  You can use either.
#
#  This builds 1 executable:
#          MATXT_CatalogSample_d - Built with the MATXT shared library
#
#####
#
CFILES1 = \
    MATXT_CatalogSample.c

OFILES1 = $(CFILES1:.c=.obj)

MATXT_HOME = c:\h\MATXT

#  The program built with Shared Library
PROG1_DLL_NAME = MATXT_CatalogSample_d

#  The informix libraries
INFORMIXDIR = c:\informix
INFORMIXLIBS = $(INFORMIXDIR)\lib\isqlt07c.lib

LIBS = $(INFORMIXLIBS) $(MATXT_HOME)\bin\MATXTAPI.lib

#  MATXT include directory
INCLUDES = -I$(MATXT_HOME)\include

# other definitions
RM = -@del /Q /F /S
LD = link

CFLAGS = -c -nologo -G5 -Od -Zi -W3 -D_DEBUG -D_MDBDLL -MD -D_MT \
        -I$(MATXT_HOME)\include -Ic:\informix\incl\esql -DSTRICT -D__STDC__=0

LDFLAGS = -nologo -debug:full -debugtype:cv -pdb:none /VERBOSE:LIB \
          -pdb:none /NODEFAULTLIB:libc.lib /NODEFAULTLIB:libcmt.lib
```

```
all: clean $(PROG1_DLL_NAME)

$(PROG1_DLL_NAME) : $(OFILES1)
    $(LD) $(LDFLAGS) $(OFILES1) \
    $(LIBS) -OUT:MATXTtestDeleteByID_d.exe
    @echo "done"

#      Clean up objects, uids, and executables
clean :
    $(RM) $(OFILES1) *.lnk *.map

clobber: clean
    $(RM) $(PROG1_DLL_NAME).exe
```

4.3 Building Application on the HP-UX Using the MATXT Library

4.3.1 Makefile for HP-UX Using the Archive MATXT Library

```
#####
#
#  makefile.hpux.a - Builds the sample MATXT_CatalogSample
#
#  Note that this makefile uses cc instead of esql for building.  You
#  can use either.
#
#  This builds 1 executable:
#          MATXT_CatalogSample_1 - Built with the MATXT static library
#
#####

CFILE1 = \
    MATXT_CatalogSample.c

OBJS1 = $(CFILE1:.c=.o)

MATXT_HOME = /h/MATXT

# The program built with Static Library
PROG1_LIB_NAME = MATXT_CatalogSample_1

# The informix libraries
INFORMIXLIBS = -L$(INFORMIXDIR)/lib -L$(INFORMIXDIR)/lib/esql \
    -lixsql -lixgen -lixos -lixassf -lixgls \
    $(INFORMIXDIR)/lib/esql/checkapi.o -lns1_s -lm -lv3 -lcl -lsec

# MATXT include directory
INCLUDES = -I$(MATXT_HOME)/include

# other definitions (must run under sh shell)

SHELL = /bin/sh
RM = /bin/rm -f
LD = cc
CFLAGS = -g -Aa -D_HPUX_SOURCE +wl +wl $(INCLUDES)
LDFLAGS = -g

all : clean $(PROG1_LIB_NAME)
```

```
$(PROG1_LIB_NAME): $(OBJS1)
    $(LD) $(OBJS1) $(LDFLAGS) -L$(MATXT_HOME)/lib -lMATXTAPI \
    $(INFORMIXLIBS) -o $(PROG1_LIB_NAME)

#      Clean up objects
clean :
    $(RM) $(OBJS1) core

#      Clean up objects and executables
clobber:
    $(RM) $(OBJS1) $(PROG1_LIB_NAME) core
```

4.3.2 Makefile for HP-UX Using the Run-Time MATXT Library

```
#####
#
#      makefile.hpux - Builds the sample MATXT_CatalogSample
#
#      Note that this makefile uses cc instead of esql for building.  You
#      can use either.
#
#      This builds 1 executable:
#          MATXT_CatalogSample_d - Built with the MATXT shared library
#
#####

CFILE1 = \
        MATXT_CatalogSample.c

OBJS1 = $(CFILE1:.c=.o)

MATXT_HOME = /h/MATXT

#  The program built with Shared Library
PROG1_DLL_NAME = MATXT_CatalogSample_d

#  The informix libraries
INFORMIXLIBS = -L$(INFORMIXDIR)/lib -L$(INFORMIXDIR)/lib/esql \
    -lixsql -lixgen -lixos -lixassf -lixgls \
    $(INFORMIXDIR)/lib/esql/checkapi.o -lnsl_s -lm -lv3 -lcl -lsec

#  MATXT include directory
INCLUDES = -I$(MATXT_HOME)/include

# other definitions (must run under sh shell)

SHELL = /bin/sh
RM = /bin/rm -f
LD = cc
CFLAGS = -g -Aa -D_HPUX_SOURCE +wl +wl $(INCLUDES)
LDFLAGS = -g -Wl,+s

all : clean $(PROG1_DLL_NAME)

$(PROG1_DLL_NAME):$(OBJS1)
    $(LD) $(OBJS1) $(LDFLAGS) -L$(MATXT_HOME)/bin -lMATXTAPI \
    $(INFORMIXLIBS) -o $(PROG1_DLL_NAME)
```

```
#      Clean up objects
clean : $(RM) $(OBJS1) core

#      Clean up objects and executables
clobber: $(RM) $(OBJS1) $(PROG1_DLL_NAME) core
```

4.4 Customizing Segments

This section is tailored out.

5 NOTES

5.1 Glossary of Acronyms

AESS	Allied Environmental Support System
API	Application Program Interface
APIRM	API Reference Manual
COE	Common Operating Environment
DII	Defense Information Infrastructure
GCCS	Global Command and Control System
IC4ISR	Integrated Command, Control, Communications, Computer, and Intelligence Surveillance Reconnaissance
IMOSS	Interim Mobil Oceanographic Support System
IP	Installation Procedures
JMCIS	Joint Maritime Command Information System
JMS	Joint METOC Segment
LLT	Latitude-Longitude-Time
MATXT	Textual Observation API Segment of the TESS(NC) METOC Database
MDXTXT	Textual Observation Database Segment of the TESS(NC) METOC Database
METOC	Meteorology and Oceanography
MIDDS	Meteorological Integrated Data Display System
NITES	Navy Integrated Tactical Environmental Subsystem

PC	Personal Computer
PM	Programming Manual
PS	Performance Specification
RDBMS	Relational Database Management System
SQL	Structured Query Language
SVD	Software Version Description
TESS(NC)	Tactical Environmental Support System Next Century